40515

National Library of Canada | Bibliothèque nationale du Canada

CANADIAN THESES ON MICROFICHE | THÈSES CANADIENNES SUR MICROFICHE

NAME OF AUTHOR/*NOM DE L'AUTEUR* Grace L. Ting

TITLE OF THESIS/*TITRE DE LA THÈSE* On Schedules With Different Release Times and Deadlines

UNIVERSITY/*UNIVERSITÉ* U. of A

DEGREE FOR WHICH THESIS WAS PRESENTED/ *GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE* M.Sc.

YEAR THIS DEGREE CONFERRED/*ANNÉE D'OBTENTION DE CE GRADE* 1979

NAME OF SUPERVISOR/*NOM DU DIRECTEUR DE THÈSE* Dr. F. Chin

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

DATED/*DATÉ* Jan 16, 1979 SIGNED/*SIGNÉ* Grace Ting

PERMANENT ADDRESS/*RÉSIDENCE FIXÉ* 26A Tai Ping Shan St.
5/F. Sheung Wan
HONG KONG

NL-91 (3-74)

National Library of Canada

Cataloguing Branch,
Canadian Theses Division

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada

Direction du catalogage
Division des thèses canadiennes

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

THE UNIVERSITY OF ALBERTA

ON SCHEDULES WITH DIFFERENT RELEASE TIMES AND DEADLINES

BY

GRACE LAI-LIN TING

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

SPRING 1979

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, the thesis entitled ON SCHEDULES WITH DIFFERENT RELEASE TIMES AND DEADLINES submitted by Grace Lai-Lin Ting in partial fulfilment of the requirements for the degree of Master of Science.

F Chen
.....
(Supervisor)

G. J. Burke
.....

Robert A. Recklow
.....

Date Jan 8 1979
.....

To My Parents

ABSTRACT

Given a set of jobs with different release times, execution times and deadlines, the problem of finding a schedule on a processor which minimizes the number of jobs failing to meet their deadlines is discussed. An optimal preemptive algorithm of time complexity $O(n^2 \log n)$ is presented for two batches of jobs of different release times. A heuristic preemptive algorithm for jobs with arbitrary release times is also found. In addition it is shown that the problem is NP-complete when preemption is not allowed and, hence, is likely to be computationally intractable.

ACKNOWLEDGEMENT

I would like to express my appreciation to Dr. F. Chin, my supervisor, for his advise, guidance and criticism throughout the preparation of this thesis.

Thanks are due to Dr. R. A. Reckhow and G. J. Butler for their careful reading and valuable suggestions. I would also like to thank Mr. Daniel Chow for his encouragement and comments.

Finally, the financial assistance received from the Department of Computing Science and National Research Council, in the form of teaching and research assistantship, is appreciated.

TABLE OF CONTENTS

CHAPTER			PAGE
I		Introduction	1
II		Properties of On-time Subsets	7
	2.1	D-schedule - Definition and Properties	8
	2.2	Properties of On-time Subsets of the Same Size	16
	2.3	Properties of On-time Subsets of Different Sizes	25
	2.4	Basic Ideas Behind the Algorithm	31
III		The Optimal Algorithm	33
	3.1	Trimming of On-time Subsets Having Non-zero Idle Times	33
	3.2	The Algorithm and Its Correctness	39
	3.3	Complexity of the Algorithm	47
IV		Particular Cases and Heuristic Algorithm ...	49
	4.1	General Model	49
	4.2	An Optimal Case	52
	4.3	Heuristic Algorithm	56

V Conclusion 60

References 62

Appendix A 64

LIST OF FIGURES

FIGURE		PAGE
1.1	Three Schedules of Example 1.1	6
2.1.1	D-schedule of a Subset S of Jobs	10
2.1.3	Schedules Before and After Interchange	14
4.1.1	A D-schedule for m Release Times	51
A	Partition of Jobs - A Non-preemptive Schedule	66

CHAPTER I

Introduction

Scheduling is the allocation of resources over time to perform a collection of jobs. General scheduling problems assume a set of resources or processors and a fixed system of jobs which is to be served by these resources. Based on prespecified properties of and constraints on the jobs and resources, the general problem is to find an efficient algorithm for constructing a schedule which optimizes or tends to optimize some desired performance measures. Two common measures are finishing time and number of late jobs. Finishing time is the time by which all the jobs are completed if they are executed according to the schedule. Late jobs are jobs completed execution after their specified deadlines. Other common measures are mean flow time, waiting time and lateness [4] but they will not be discussed in this thesis.

A schedule is an arrangement of the jobs to be executed on the processors. A preemptive schedule is a schedule in which one may suspend the execution of a job before its completion and resume its execution at a later time. An on-time schedule is one in which all jobs can be completed before their deadlines. A set of jobs is said to be on-time if there is a schedule in which all the jobs can be

completed before their deadlines. Release time of a job is the time at which the job is released for execution. We denote the release time, deadline and execution time of job i by r(i), d(i) and t(i) respectively.

Much effort has been spent on the problem of determining whether there exists a schedule in which each job is completed before its deadline. Garey and Johnson [5,6] have considered non-preemptive scheduling algorithms on two processors for a set of n jobs with precedence constraints. The earlier paper [5] describes an algorithm which finds an on-time schedule whenever one exists for the problem in which each job has unit execution time and zero release time (i.e. t(i)=1 and r(i)=0). The algorithm takes either $O(n^2)$ or O(n**log(7)) time, depending on whether or not the precedence relation on the jobs is given in transitively closed form. They [6] further relax the constraint on job release times, allowing them to be arbitrary integers and give an $O(n^3)$ algorithm which produces an on-time schedule whenever one exists. Blazewicz [3] generalizes the algorithm [5] for the case of arbitrary job execution times, allowing job preemption. Recently, Simons [12] has developed an $O(n^2 \log n)$ recursive algorithm for the problem in which jobs have the same execution time but arbitrary release times. It produces an on-time non-preemptive schedule on one processor with the earliest possible completion time whenever one exists.

The problems described in the previous paragraph are basically the problems of determining whether there exists a schedule in which each job is completed before its deadline and finding such a schudule whenever one exists. The problem of finding a schedule which minimizes the number of late jobs for a given set of jobs is more practical and is of current interest. We call a schedule which achieves this objective an optimal schedule. Moore [10] has found an algorithm for the problem on one processor in which each job is associated with an execution time t(i), a deadline d(i) and all the jobs are released for execution at the same time (i.e. r(i)=0). Sidney [11] further extends the result to include the condition that there is a subset of jobs which must be completed on time in the schedule. Both algorithms can be implemented in O(n log n) time.

It is hoped that the result can be generalized to the case in which each job is associated with one more parameter r(i), the release time; i.e. each job can only be executed after it has been released. However the problem of minimizing the number of late jobs is still an open problem even for one processor. A particular case of this has been considered by Kise, Ibaraki and Mine [9]. They give an $O(n^2)$ non-preemptive algorithm under the restrictive assumption that $r(i) < r(j)$ implies $d(i) \leq d(j)$.

In this thesis we study the problem of minimizing the number of late jobs for the case in which each job is associated with an executing time t(i), a deadline d(i) and a release time r1 or r2 (i.e. r(i) = r1 or r2, r1<r2). Only one processor is considered. An $O(n^2 \log n)$ time algorithm which produces an optimal schedule is found under the assumption that preemption is allowed. The problem is NP-complete (unlikely to be solved in polynomial time) if preemption is not allowed [Appendix A] and, hence, is likely to be computationally intractable.

In Chapter II, we define a necessary and sufficient condition for a set of jobs to be on-time and show some properties of on-time subsets of jobs. In Chapter III, we present an $O(n^2 \log n)$ algorithm which gives a preemptive schedule that minimizes the number of late jobs. We also prove the algorithm is correct. In Chapter IV, we study the particular case in which the set of jobs released at times later than the earliest release time is on-time and find an optimal algorithm of $O(n \log n)$ time. We also extend this algorithm to a heuristic algorithm for the problem with arbitrary release times. The number of late jobs produced by the heuristic algorithm is not more than m times the optimal, where m is the number of distinct release times.

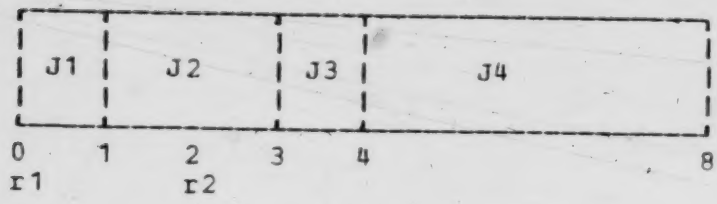
A simple example of the problem is given below.

Example 1.1

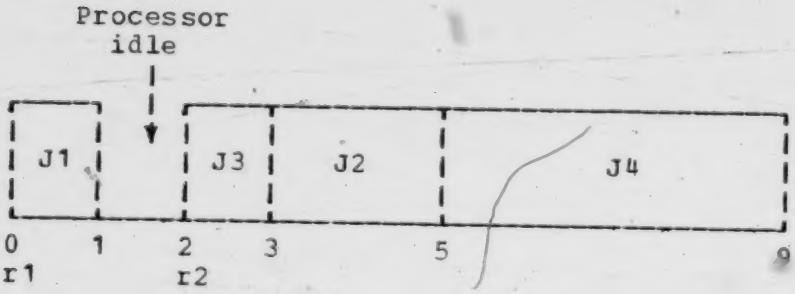
Consider a set of 4 jobs with two release times, 0 and 2. (i.e. r(i) = r1 or r2; r1=0, r2=2)

		Job			
		J1	J2	J3	J4
release time	r(i)	0	0	2	2
execution time	t(i)	1	2	1	4
deadline	d(i)	1	4	3	6

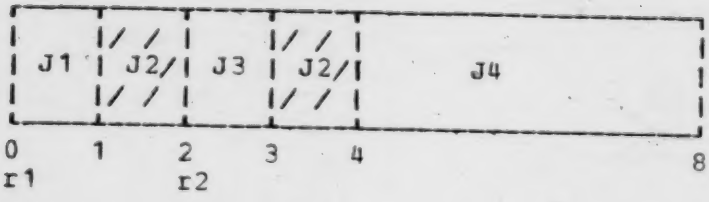
The schedules shown in Figures 1.1(a) and 1.1(b) both result in two late jobs, while the optimal schedule shown in Figure 1.1(c) gives only one late job. Note that the optimal schedule in Figure 1.1(c) is a preemptive schedule.



(a) A schedule with late jobs {J3,J4}



(b) A schedule with late jobs {J2,J4}



(c) An optimal schedule with late job {J4}

Figure 1.1 : Three Schedules of Example 1.1

CHAPTER II

Properties of On-time Subsets

In this chapter we study the problem of minimizing the number of late jobs in scheduling a given set of jobs on one processor; each job is associated with an execution time t(i), a deadline d(i) and a release time either r1 or r2 (r1 < r2). From now on, the discussion is restricted to a set of n jobs, J = {1, 2, ..., n}, released either at r1 or r2. Without loss of generality, we assume r1=0, and $d(1) \leq d(2) \leq \ldots \leq d(n)$.

The algorithm for the problem partitions the set of jobs into two subsets, an on-time subset and a subset of late jobs. The partition produces an optimal schedule if the jobs in the on-time subset are arranged according to the D-schedule (to be defined in Section 2.1), and are followed by the jobs in the second set in any order.

In Section 2.1 , we give a special preemptive scheduling algorithm which produces an on-time schedule for an <u>on-time</u> subset of jobs. We also prove a necessary and sufficient condition for a subset of jobs to be on-time. In Section 2.2, we study some properties of on-time subsets of jobs and define a performance measure to compare two on-time subsets. We will use the results from Section 2.2 and Section 2.3 to determine which jobs to be deleted in the

case where some jobs must be late in the schedule. Finally, in Section 2.4 we discuss the basic ideas behind the algorithm.

2.1 D-schedule - Definition and Properties

We first define a special preemptive schedule called D-schedule for scheduling the jobs. We show in Lemma 2.1.2 that a subset S of jobs is on-time if and only if S is on-time when its jobs are executed according to the D-schedule. Based on this result, it is clear that the search for an optimal schedule for the problem can be restricted to schedules such that all on-time subsets of jobs are executed according to D-schedule. We then give a necessary and sufficient condition for a subset of jobs to be on-time in Theorem 2.1.3.

Let

E1 and E2 be subsets of jobs released at r1 and r2 respectively.

Then the union of E1 and E2 is J,

$E1 \cup E2 = J = \{1, 2, \ldots, n\}$.

Definition 2.1.1

A D-schedule of a subset S of jobs is a schedule which satisfies the following two conditions :

(1) All jobs in $S \cap E1$ (intersection of S and E1) are

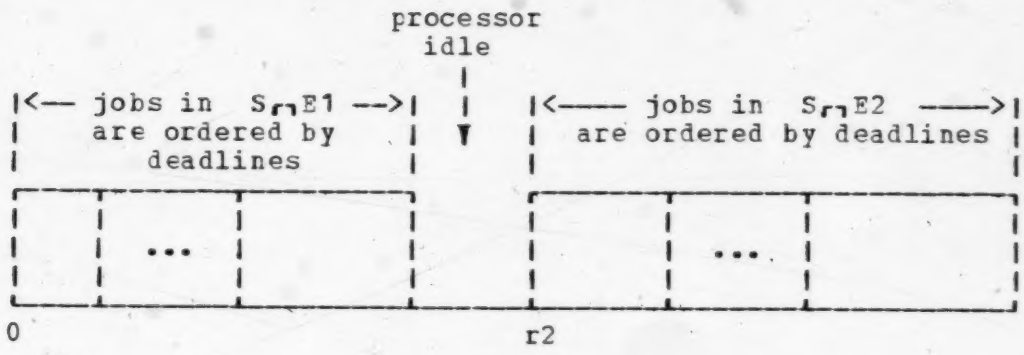
executed according to the order of non-decreasing deadlines. If d(i)=d(j) and i<j then Job i is executed before Job j.

(2) All jobs in S (including, if any, the remaining portion of the job which has been started execution before r2 but not yet completed at r2) scheduled after r2 are executed according to the order of non-decreasing deadlines. If d(i)=d(j) and i<j then Job i is executed before Job j. ▪

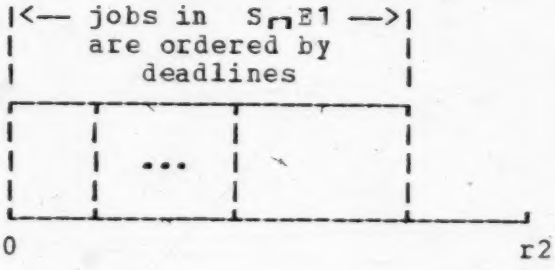
According to condition (1) of the definition, the jobs scheduled between r1 and r2 are all in E1 and are executed according to the order of non-decreasing deadlines. If all the jobs in $S \cap E1$ can be completed execution by r2, then the processor is idle until r2; and the jobs executed after r2 will be all in $S \cap E2$. If, on the other hand, some jobs in $S \cap E1$ have to be executed after r2, a job in E1 may be preempted at r2 and the remaining portion of that job together with all the jobs left behind will be scheduled for execution after r2 according to deadlines.

Figure 2.1.1 illustrates pictorially the D-schedule of a subset S of jobs. It can be shown that the D-schedule of S yields the earliest possible finishing time among all schedules of S.

The following lemma is used to prove Lemma 2.1.2 which

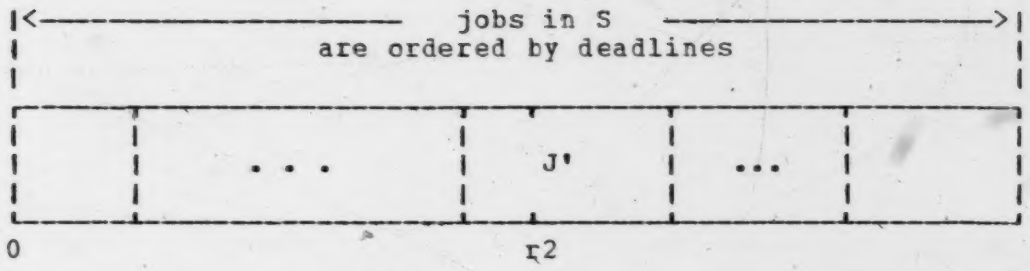


(a) $\sum_{j \in S_{\cap} E1} t(j) \leq r2$ and $S_{\cap} E2 \neq \phi$

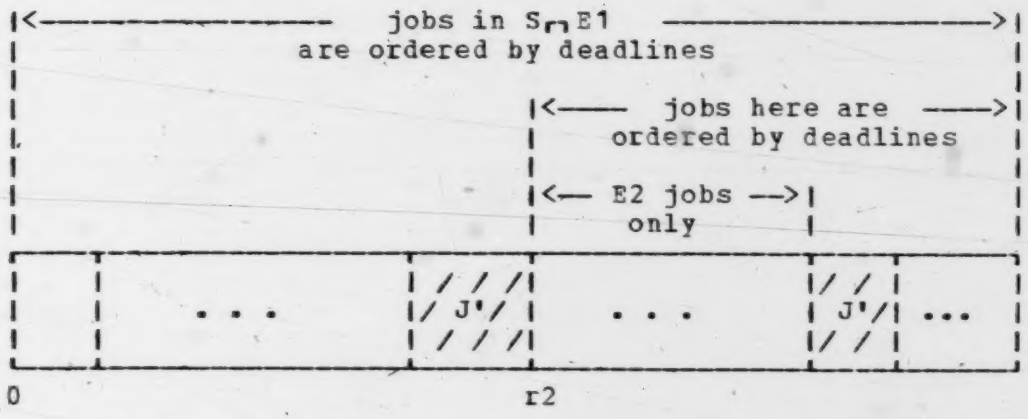


(b) $\sum_{j \in S_{\cap} E1} t(j) \leq r2$ and $S_{\cap} E2 = \phi$

Figure 2.1.1 : D-schedule of a Subset S of Jobs



(c) $\sum_{j \in S \cap E1} t(j) > r2$, J' is not preempted



(d) $\sum_{j \in S \cap E1} t(j) > r2$ and J' is preempted

Figure 2.1.1 continued

states that a subset S of jobs is on-time if and only if the D-schedule of S is on-time.

Lemma_2.1.1 (Jackson [7])
Let S' be any on-time subset of jobs released at the same time. If the jobs in S' are ordered by non-decreasing deadlines, then the resulting schedule will be on-time. ▪

Lemma_2.1.2
S is an on-time subset of jobs if and only if the D-schedule of S is on-time.

Proof
(<=) obvious.
(=>)
Consider an on-time schedule (preemptive or non-preemptive) of S which is not a D-schedule. An on-time D-schedule of S can be obtained by the following algorithm :

step 1. rearrange separately the jobs (and segments of the jobs) scheduled after r2 and those scheduled between r1 and r2 by non-decreasing deadlines; and combine segments of the same job.

step 2. Let J' be the job (or a segment of a preempted job) scheduled just before r2; and J" be the first job (or a segment of a preempted job) in $S_{\cap} E1$ scheduled at or after r2.

If $d(J') < d(J'')$, or if $d(J') = d(J'')$ and $J' < J''$, STOP. (The schedule is a D-schedule). Otherwise go to step 3.

step 3. Interchange J' and J'' if $t(J') = t(J'')$ otherwise interchange the smaller job with an equal portion of the larger job. Return to step 1.

The algorithm halts after a finite number of steps.

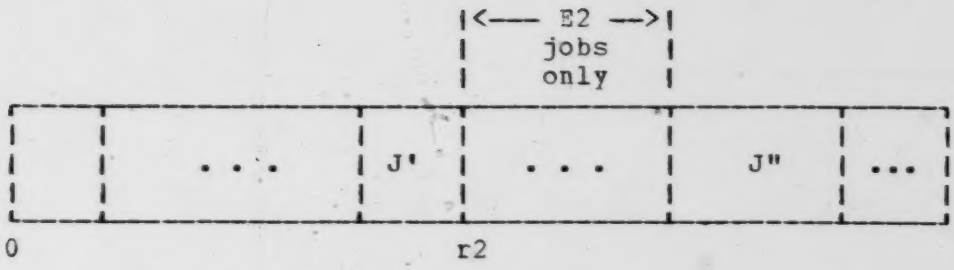
By Lemma 2.1.1, step 1 will result in an on-time schedule. The job interchange is illustrated in Figure 2.1.3 for the case where $t(J'') > t(J')$. The figure for the case where $t(J'') \leq t(J')$ is similar and is omitted. Clearly, all jobs will be on-time after the interchange. Hence the above algorithm produces an on-time D-schedule of S. ∎

Based on Lemmam 2.1.2, it is clear that the search for an optimal schedule can be restricted to D-schedules.

The <u>finishing time</u> $F(S)$ of the D-schedule of a non-empty subset S of jobs is the time by which all jobs in S are completed.

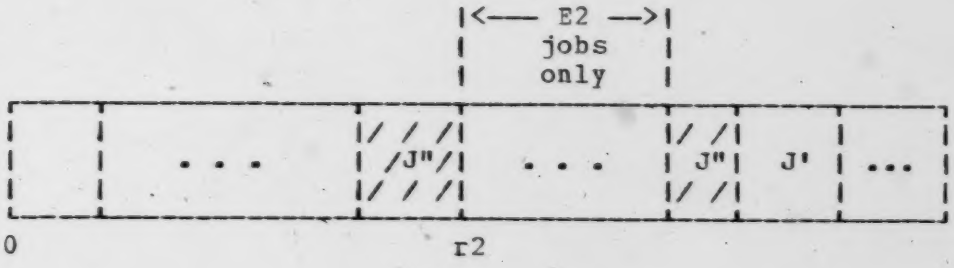
$F(S)$ can be computed by the following equation.

$$F(S) = \begin{cases} \sum_{j \in S} t(j) & \text{if } \sum_{j \in S \cap E1} t(j) > r2 \text{ or } S \cap E2 = \phi \\ r2 + \sum_{j \in S \cap E1} t(j) & \text{if } \sum_{j \in S \cap E1} t(j) \leq r2 \text{ and } S \cap E2 \neq \phi \end{cases}$$



(a) Schedule Before Interchange

d(J') > d(J"); or d(J') = d(J") but J' > J"



(b) Schedule After Interchange

Figure 2.1.3 : Schedules Before and After Interchange for the case t(J") > t(J')

The idle time I(S) of the D-schedule of a non-empty subset S of jobs is the period when the processor remains idle. When all the E1 jobs in S complete execution before r2, the processor will remain idle until r2 and then execute the E2 jobs in S. I(S) can be computed by the following equation.

$$I(S) = \max\{0, r2 - \sum_{j \in S \cap E1} t(j)\}.$$

We say that S has idle time if I(S) > 0.

Theorem 2.1.3

S = {1, 2, ..., p} is an on-time subset of jobs if and only if

$F(\{1, 2, \ldots, i\}) \leq d(i) \qquad i=1, \ldots, p.$

(N.B. $d(1) \leq d(2) \leq \ldots \leq d(p)$)

Proof

(<=) obvious.

(=>)

The proof is by contradiction. Assume S is on-time but $F(\{1,2, \ldots, i'\}) > d(i')$ for some $1 \leq i' \leq p$. Then in the D-schedule of {1,2, ..., i'}, there is a job, say Job j, completed after d(i'), in other words after d(j). This contradicts the assumption that S is on-time. ▪

The above theorem gives a necessary and sufficient

condition for a subset of jobs to be on-time. We will use this condition to check if a subset of the jobs is on-time in the algorithm to be presented in Chapter III.

2.2 Properties of On-time Subsets of the Same Size

In Section 2.1, we have derived a necessary and sufficient condition for a subset of jobs to be on-time. When a subset S of jobs fails to meet the condition we know that there does not exist any on-time schedule for S. In this section and in Section 2.3, we study the problem of choosing some on-time subsets such that one of them will later guarantee a maximal on-time subset for the set of jobs, J. We now give the notion of 'Late' as the performance measure for comparing two on-time subsets S1 and S2. From now on let us assume T to be a set of jobs with deadlines greater than or equal to the deadline of any job in $S1 \cup S2$, and $T \cap (S1 \cup S2) = \phi$; where S1 and S2 are two on-time subsets whose performance measures are to be compared.

Definition 2.2.1

Late[S] is the minimum number of late jobs of S among all possible schedules of S. ∎

A maximal or optimal on-time subset of S is an on-time subset which has the maximal number of jobs.

Our goal is to minimize the number of late jobs; equivalently, it is tc find a schedule of S which has Late[S] late jobs, i.e. produces a maximal on-time subset of S. Of course, Late[S] = 0 if and only if S is on-time. We call a schedule of S an optimal schedule if it has Late[S] late jobs.

Definition 2.2.2

S1 and S2 are two on-time subsets of jobs.
S1 is said to be better than S2 (more precisely, "at least as good as") if, for any T as described above,

$$|S1 \sqcup T| - Late[S1 \sqcup T] \geq |S2 \sqcup T| - Late[S2 \sqcup T]$$

or equivalently,

$$Late[S1 \sqcup T] \leq Late[S2 \sqcup T] + |S1| - |S2| \quad .$$

Thus, to search for an optimal on-time subset for the problem from two on-time subsets of the jobs, one can just keep the better subset and ignore the other. Theorem 2.2.3 and Theorem 2.2.4 give sufficient conditions for one on-time subset to be better than the other. Before we state the two theorems, we first prove the following two lemmas.

Lemma 2.2.1

Let S1 and S2 be any two subsets of jobs (not necessarily on-time)
If

$$\sum_{j \in S1} t(j) \leq \sum_{j \in S2} t(j)$$

and $F(S1) \leq F(S2)$,

Then

$$F(S1 \sqcup T) \leq F(S2 \sqcup T)$$

Proof.

If $S1 \sqcup T$ has no idle time,

$$F(S1 \sqcup T) = \sum_{j \in S1 \sqcup T} t(j)$$

$$\leq \sum_{j \in S2 \sqcup T} t(j)$$

$$\leq F(S2 \sqcup T) .$$

If $S1 \sqcup T$ has idle time (the idle time must be before r2),

$$F(S1 \sqcup T) = r2 + \sum_{j \in S1 \cap E2} t(j) + \sum_{j \in T \cap E2} t(j)$$

$$= F(S1) + \sum_{j \in T \cap E2} t(j)$$

$$\leq F(S2) + \sum_{j \in T \cap E2} t(j)$$

$$\leq F(S2 \sqcup T). \quad \blacksquare$$

If S1 and S2 are on-time subsets of jobs and $S2 \sqcup T$ is also on-time, the following lemma gives sufficient conditions for $S1 \sqcup T$ to be on-time.

Lemma 2.2.2

Let S1 and S2 be any two on-time subsets of jobs such that

(1) $\sum_{j \in S1} t(j) \leq \sum_{j \in S2} t(j)$

and

(2) $F(S1) \leq F(S2)$.

Then

$$Late[S2 \sqcup T] = 0 \quad \Rightarrow \quad Late[S1 \sqcup T] = 0$$

Proof.

Without loss of generality, let

$$T = \{T1, T2, \ldots, Tk\} \qquad k \geq 1.$$

and assume

$$d(T1) \leq d(T2) \leq \ldots \leq d(Tk) .$$

If $Late[S2 \sqcup T] = 0$, i.e. $S2 \sqcup T$ is on-time,

by Theorem 2.1.3, for any $i \leq k$

$F(S2 \sqcup \{T1, T2, \ldots, Ti\}) \leq d(Ti)$.

By Lemma 2.2.1,

$$F(S1 \sqcup \{T1, T2, \ldots, Ti\}) \leq F(S2 \sqcup \{T1, T2, \ldots, Ti\}) \leq d(Ti) .$$

Hence $S1 \sqcup T$ is on-time or $Late[S1 \sqcup T] = 0$. ∎

The following two theorems give sufficient conditions for S1 to be better than S2, where S1 and S2 differ by one job.

Theorem 2.2.3

Let S, S1 and S2 be on-time subsets of jobs, where

$$S1 = S \sqcup \{x\}$$

and $S2 = S \sqcup \{y\}$,

with either both $x, y \in E1$ or $E2$;

or ($x \in E1$ and $y \in E2$).

If

$$t(x) \leq t(y) \quad \left(\text{i.e. } \sum_{j \in S1} t(j) \leq \sum_{j \in S2} t(j) \right)$$

then

S1 is better than S2,

i.e. $Late[S1 \sqcup T] \leq Late[S2 \sqcup T]$.

Proof

Suppose $Late[S2 \sqcup T] = v$ and the late jobs are $\{w1, w2, \ldots, wv\}$.

Let

$A1 = \{w1, w2, \ldots, wv\} \sqcap S2$,

$A2 = \{w1, w2, \ldots, wv\} \sqcap T$.

$A1 \sqcup A2 = \{w1, w2, \ldots, wv\}$ (N.B. A1 or A2 may be empty),

and $(S2 \sqcup T) - (A1 \sqcup A2)$ is on-time.

Case (i) $y \in A1$

Without loss of generality assume $y = w1$.

Clearly

$$(S1 \sqcup T) - \{x, w2, \ldots, wv\}$$
$$= (S2 \sqcup T) - \{y, w2, \ldots, wv\}$$

is on-time.

Therefore $Late[S1 \sqcup T] \leq v$.

Case (ii) $y \neg\in A1$

Then

$$S1-A1 = (S-A1) \sqcup \{x\},$$

$$S2-A1 = (S-A1) \sqcup \{y\}.$$

Since $t(x) \leq t(y)$

and either both $x,y \in E1$ or $E2$; or $x \in E1$ and $y \in E2$

we have

$$F(S1-A1) \leq F(S2-A1) .$$

Thus (S1-A1) and (S2-A1) satisfy the conditions of Lemma 2.2.2. Hence by Lemma 2.2.2,

Late[$(S2-A1) \sqcup (T-A2)$] on-time

=> Late[$(S1-A1) \sqcup (T-A2)$] on-time.

Therefore Late[$S1 \sqcup T$] $\leq v$. ∎

Theorem 2.2.4

Let S, S1 and S2 be on-time subsets of jobs, where

$$S1 = S \sqcup \{x\},$$

and $S2 = S \sqcup \{y\}$,

with $x \in E2$ and $y \in E1$ and y is the largest job in $S2 \sqcap E1$ (i.e. y has the largest execution time amongst the jobs in $S2 \sqcap E1$).

If

$$t(x) \leq t(y) \quad (\text{ i.e. } \sum_{j \in S1} t(j) \leq \sum_{j \in S2} t(j))$$

and

$$F(S1) \leq F(S2)$$

then

S1 is better than S2,

i.e. $Late[S1 \sqcup T] \leq Late[S2 \sqcup T]$.

Proof.

Suppose $Late[S2 \sqcup T] = v$ and the late jobs are $\{w1, w2, \ldots, wv\}$.

Let

$A1 = \{w1, w2, \ldots, wv\} \sqcap S2$,

$A2 = \{w1, w2, \ldots, wv\} \sqcap T$.

$A1 \sqcup A2 = \{w1, w2, \ldots, wv\}$ (N.B. A1 or A2 may be empty),

and $(S2 \sqcup T) - (A1 \sqcup A2)$ is on-time.

Case (i) $y \in A1$

Without loss of generality assume $y=w1$.

Clearly

$$(S1 \sqcup T) - \{x, w2, \ldots, wv\}$$
$$= (S2 \sqcup T) - \{y, w2, \ldots, wv\}$$

is on-time.

Therefore $Late[S1 \sqcup T] \leq v$.

Case (ii) $y \neg\in A1$

(a) $A1 \sqcap E1 \neq \phi$

Let $y' \in A1 \sqcap E1$.

As $t(y') \leq t(y)$, from Theorem 2.2.3 we have

$$\text{Late}[(S2-\{y\}) \sqcup T] \leq \text{Late}[(S2-\{y'\}) \sqcup T]$$
$$= v-1.$$

Since $S1-\{x\} = S2-\{y\}$,

$$\text{Late}[(S1-\{x\}) \sqcup T] \leq v-1$$

$$\Rightarrow \quad \text{Late}[S1 \sqcup T] \leq v.$$

(b) $A1 \cap E1 = \phi$ (i.e. A1 is a subset of E2)

Consider S1-A1 and S2-A1.

$$S1-A1 = (S-A1) \sqcup \{x\},$$
$$S2-A1 = (S-A1) \sqcup \{y\}.$$

$$\sum_{j \in S1-A1} t(j) \leq \sum_{j \in S2-A1} t(j) \qquad (\text{since } t(x) \leq t(y)).$$

Since $A1 = \phi$ or all jobs in A1 are E2 jobs;

and $x \in (S1-A1) \cap E2$,

$$F(S1) \leq F(S2)$$

$$\Rightarrow \quad F(S1-A1) \leq F(S2-A1).$$

By Lemma 2.2.2,

$$(S2-A1) \sqcup (T-A2) \quad \text{on-time}$$

$$\Rightarrow \quad (S1-A1) \sqcup (T-A2) \quad \text{on-time.}$$

Therefore $\text{Late}[S1 \sqcup T] \leq v$.

This completes the proof. ∎

Both Theorem 2.2.3 and Theorem 2.2.4 give sufficient conditions for one on-time subset of jobs to be better than the other. Hence whenever there are two on-time subsets of

the jobs satisfying the conditions of either one of these two theorems and it is given that one of them will derive an optimal solution for the problem, we can restrict the search for an optimal schedule to schedules derived from the better subset.

Let us further analyse Theorem 2.2.3 and Theorem 2.2.4 . S1 and S2 are two on-time subsets which differ by one job. Let

$S1 = S \sqcup \{x\}$,

$S2 = S \sqcup \{y\}$ for some subset S of jobs.

Consider the following four cases for the pair of jobs x and y.

(1) $x, y \in E1$

(2) $x, y \in E2$

(3) $x \in E1$, $y \in E2$

(4) $x \in E2$, $y \in E1$

and y is the largest the job in $S2 \cap E1$.

Under the first three cases, if $t(x) \leq t(y)$, by Theorem 2.2.3 we can then conclude that S1 is better than S2. However under case (4) (the case considered in Theorem 2.2.4), it may happen that $t(x) < t(y)$ but $F(S1) > F(S2)$. Under such a case, we cannot draw any conclusion about S1 and S2 but have to retain both subsets for the search for an optimal solution. An example of this is given below.

Example 2.2.1

		Job		
		J1	J2	J3
release time	r(i)	0	0	2
execution time	t(i)	1	2	1.5
deadline	d(i)	1	3	3.5

S1={1,3} and S2={1,2} are two on-time subsets of the jobs. In this example, t(3) < t(2) but F(S1) > F(S2).

Consider J4 : r(4)=0, t(4)=1, d(4)=3.5
Clearly S1⊔{4} is on-time but S2⊔{4} is not.
However with J4 : r(4)=2, t(4)=0.5, d(4)=3.5 ,
S2⊔{4} is on-time but S1⊔{4} is not. ∎

2.3 Properties of On-time Subsets of Different Sizes

In Section 2.2., we have studied the performance of two on-time subsets of jobs of the same size and have given in Theorem 2.2.3 and Theorem 2.2.4 sufficient conditions for one subset to be better than the other. In this section we study the performance of two on-time subsets of different sizes. We give sufficient conditions (in Theorem 2.3.2) for one subset to be better than the other.

The following lemma tells us how to choose a job to be deleted so that the resulting set is on-time. It will be used to prove Theorem 2.3.2 .

Lemma 2.3.1

Let S be an on-time subset of jobs and let k be a job not in S with deadline greater than or equal to the deadline of any job of S.

Suppose $S \cup \{k\}$ is not on-time. Then

(a) If $S \cup \{k\} - \{x\}$ is on-time for some $x \in (S \cup \{k\}) \cap E1$ then $S \cup \{k\} - \{y\}$ is on-time for all $y \in S \cup \{k\}$ with $t(y) \geq t(x)$.

(b) If $S \cup \{k\} - \{x\}$ is on-time for some $x \in (S \cup \{k\}) \cap E2$ then $S \cup \{k\} - \{y\}$ is on-time for all $y \in S \cup \{k\} \cap E2$ with $t(y) \geq t(x)$.

Proof

Let

$$S' = S \cup \{k\} - \{x,y\},$$

$$S1 = S \cup \{k\} - \{y\} = S' \cup \{x\},$$

$$S2 = S \cup \{k\} - \{x\} = S' \cup \{y\}.$$

(a) $x \in (S \cup \{k\}) \cap E1$

$t(x) \leq t(y)$ and $x \in E1$

=>

$$\sum_{j \in S1} t(j) \leq \sum_{j \in S2} t(j)$$

and $F(S1) \leq F(S2)$.

By Lemma 2.2.2,

S2 on-time => S1 on-time.

(b) $x \in (S \sqcup \{k\}) \sqcap E2$

The proof is similar to case (a) ▪

Whenever a subset S of the jobs is on-time except the last job (the job with the latest deadline), one can either delete one or a subset of jobs from S to make the resulting set on-time. It is shown in the following theorem that deleting one single job will always result in a better set than deleting a subset of jobs.

Theorem 2.3.2

Let S be an on-time subset of jobs and let k be a job not in S with deadline greater than or equal to the deadline of any job of S.

Suppose $S \sqcup \{k\}$ is not on-time.

If $S \sqcup \{k\}-X$ is on-time, where X is a subset of $S \sqcup \{k\}$ and $|X|>1$, but $S \sqcup \{k\}-X'$ is not on-time for any proper subset X' of X; and $S \sqcup \{k\}-\{y\}$ is on-time, where y is the largest job in either $(S \sqcup \{k\}) \sqcap E1$ or $(S \sqcup \{k\}) \sqcap E2$,

then

$S \sqcup \{k\}-\{y\}$ is better than $S \sqcup \{k\}-X$

i.e. $\text{Late}[(S \sqcup \{k\}-\{y\}) \sqcup T] \leq$

$\text{Late}[(S \sqcup \{k\}-X) \sqcup T] + |X| - 1$.

Proof

Let

$X = \{x1, x2, \ldots, xp\}, \quad p>1.$

Case(i) $y \in E1$

It can be shown by contradiction that

$$t(xi) < t(y) \quad \text{for all } xi \in X.$$

If not, by Lemma 2.3.1(a), $S \sqcup \{k\} - \{xi\}$ would be on-time. This contradicts the assumption that $S \sqcup \{k\} - X'$ is not on-time for any proper subset X' of X.

Without loss of generality assume $x1 \in E1$ if $X \cap E1 \neq \phi$; otherwise all the xi's are in E2.

Let

$$S' = S \sqcup \{k\} - \{y, x1, x2, \ldots, xp\},$$

$$S1 = S \sqcup \{k\} - \{y, x2, \ldots, xp\} = S' \sqcup \{x1\}$$

$$S2 = S \sqcup \{k\} - \{x1, x2, \ldots, xp\} = S' \sqcup \{y\}$$

(a) If $x1 \in E1$ (i.e. $X \cap E1 \neq \phi$), from Theorem 2.2.3, we have

$$Late[S1 \sqcup T] \leq Late[S2 \sqcup T] .$$

But

$$\begin{aligned} Late[(S \sqcup \{k\} - \{y\}) \sqcup T] &\leq Late[S1 \sqcup T] + |X| - 1 \\ &\leq Late[S2 \sqcup T] + |X| - 1 \\ &\leq Late[(S \sqcup \{k\} - X) \sqcup T] + |X| - 1 . \end{aligned}$$

(b) If all jobs in X are E2 jobs,

Since $t(x1) < t(y)$, we have

$$\sum_{j \in S1} t(j) < \sum_{j \in S2} t(j)$$

Consider

$S1' = S \sqcup \{k\} - \{y\}$,

$S2' = S \sqcup \{k\} - \{x1\}$.

S1' on-time but S2' not on-time

$\Rightarrow$ $F(S1') < F(S2')$.

Since all the xi's are E2 jobs, we have

$F(S1) < F(S2)$.

From Theorem 2.2.4, the result follows.

Case (ii) $y \in E2$

(a) $X \cap E2 \neq \phi$ (X has a E2 job)

The proof is similar to Case (i)(a).

(b) $X \cap E2 = \phi$ (all jobs in X are E1 jobs)

$(S \sqcup \{k\}) - X$ is on-time. This implies that there are some jobs in E1 (or the remaining portion of a job which has been preempted at r2) scheduled after r2 when $S \sqcup \{k\}$ is arranged by D-schedule.

Let x1 have the least execution time amongst the jobs in X. $S \sqcup \{k\} - \{x1\}$ does not result in an idle time otherwise successive deletion of E1 jobs will never produce an on-time set. It is clear that $t(x1) < t(y)$, or else $S \sqcup \{k\} - \{x1\}$ would be on-time, contradicting the assumption that $S \sqcup \{k\} - X'$ is not on-time for any proper subset X' of X.

Let

$S' = S \sqcup \{k\} - \{y, x1, x2, \ldots, xp\}$,

$S1 = S \sqcup \{k\} - \{y, x2, \ldots, xp\} = S' \sqcup \{x1\}$ $x1 \in E1$,

$$S2 = S \cup \{k\} - \{x1, x2, \ldots, xp\} = S' \cup \{y\} \quad y \in E2.$$

Since $x1 \in E1$ and $t(x1) < t(y)$,
from Theorem 2.2.3, the result immediately follows.

This completes the procf. •

Hence whenever the last job (the job with the latest deadline) of a subset S of jobs fails to meet its deadline in the D-schedule, we need only to consider those on-time subsets which result from deleting one single job from S. Furthermore by Theorem 2.2.3, it is sufficient to consider only the two on-time subsets which result respectively from deleting the largest job of $S \cap E1$ and $S \cap E2$ (based on Lemma 2.3.1 one of these two subsets must be on-time) because according to Theorem 2.2.3 they are respectively better than any other on-time subset resulting from deleting some other job from $S \cap E1$ and $S \cap E2$. Moreover by Thoerem 2.2.3 or Theorem 2.2.4, depending on which subset has shorter execution time, one of these two subsets can be eliminated if they satisfy the conditions of the theorem.

To summarize, from the subset S, one can reduce the search for an optimal solution to <u>at most two</u> on-time subsets of S from which an optimal solution of the problem may be found.

2.4 Basic Ideas Behind the Algorithm

We now discuss briefly how the algorithm for the case of two release times works. The jobs in J = {1,2,...,n} are scheduled according to D-schedule. Whenever there is a job failing to meet its deadline, from Lemma 2.1.1, we see that no matter how the jobs in the current D-schedule are arranged, at least one of them must be late. Hence a job from the current D-schedule has to be deleted. In the case where all jobs have the same release time, Moore [10] has proved that deleting the job with the largest execution time, or the largest job in short, is sufficient to guarantee an optimal schedule. However in our model there are two batches of jobs, E1 and E2 released at r1 and r2 respectively. One may think of deleting the largest job from E1, E2 or from the set of jobs that have been scheduled so far will be a sufficient criterion. Unfortunately it turns out that none of these choices guarantees an optimal schedule.

Example 2.2.1 illustrates that the choice of the job to be deleted depends on the jobs that have not yet been scheduled. Thus whenever there is a job failing to meet its deadline, instead of considering only the on-time subset resulting from deleting the largest job from either E1 or E2, we keep both on-time subsets resulting from deleting the largest job from E1 and E2 respectively. In Section 2.2 and Section 2.3 we have studied why it is sufficient to consider

only these two on-time subsets. If only one of these two subsets is on-time, we just consider the on-time one in the search for an optimal solution. If both subsets are on-time and they satisfy the conditions of Theorem 2.2.3 or Theorem 2.2.4, it has been shown that one of these two on-time subsets can be eliminated; otherwise we have to keep both subsets in the search for an optimal schedule. In such a case, it is shown in Section 3.1 that one of these two on-time subsets has idle time.

For each of the on-time subsets (at most two) as described above, we carry on scheduling the rest of the jobs. After the k-th job has been scheduled, in the worst case, it seems that there are 2**(k-1) cases. However all those on-time subsets having idle times are shown to be no better than a particular subset, M(k), which is defined in Section 3.1. Such subsets can be replaced by M(k), thus reducing the number of subsets involved in the search for an optimal schedule from 2**(k-1) to k in the worst case. These are the ideas that lead to a polynomial algorithm for the problem.

CHAPTER III

The Optimal Algorithm

In Chapter II we showed that if a subset S of the jobs has one late job, then the search for an optimal solution for the problem can be restricted to at most two on-time subsets of S obtained respectively by deleting the largest job from $S \cap E1$ and $S \cap E2$. When the k-th job is considered, in the worst case there would be 2**(k-1) on-time subsets involved in the search for an optimal solution for the problem. However some of these subsets can be trimmed off, thus lowering the time complexity to polynomial.

In Section 3.1, we show how some of these on-time subsets can be trimmed off. In Section 3.2, we present an $O(n^2 \log n)$ algorithm and prove its correctness.

3.1 Trimming of On-time Subsets Having Non-zero Idle Times

Let $N(k) = \{1, 2, \ldots, k\}$ be the set of the first k jobs of J, $1 \le k \le n$. Note that $d(1) \le d(2) \le \ldots \le d(k)$. Let S be an on-time subset of N(k-1). Suppose $S \cup \{k\}$ is not on-time and suppose S1 and S2 are the on-time subsets of $S \cup \{k\}$ obtained respectively by deleting the largest job from $(S \cup \{k\}) \cap E1$ and $(S \cup \{k\}) \cap E2$. The following lemma shows that if S1 and S2 do not satisfy the conditions of

Theorem 2.2.4 (in such a case we cannot tell which is better), then S1 has idle time (defined in Section 2.1) but S2 does not.

Lemma 3.1.1

Let S be an on-time subset of N(k-1). Suppose $S \sqcup \{k\}$ is not on-time.

Let x and y be jobs such that

$$t(y) = \max\{t(j) : j \in (S \sqcup \{k\}) \cap E1\},$$

$$t(x) = \max\{t(j) : j \in (S \sqcup \{k\}) \cap E2\}.$$

Let

$$S' = S \sqcup \{k\} - \{x,y\}.$$

Suppose both $S1 = S' \sqcup \{x\}$ and $S2 = S' \sqcup \{y\}$ are on-time. (N.B. From Lemma 2.3.1, one of these two subsets must be on-time)

If

$$\sum_{j \in S1} t(j) < \sum_{j \in S2} t(j) \quad \text{(i.e. } t(x) < t(y)\text{)}$$

but

$$F(S1) > F(S2)$$

then $I(S1) > 0$ but $I(S2) = 0$.

Proof

$$S1 = S \sqcup \{k\} - \{y\} = S' \sqcup \{x\} \qquad x \in E2,$$

$$S2 = S \sqcup \{k\} - \{x\} = S' \sqcup \{y\} \qquad y \in E1.$$

Since both S1 and S2 are on-time and $(S \sqcup \{k\}) \cap E2$ is non-

empty, the total execution time of the jobs in $(S \cup \{k\}) \cap E1$ must be greater than r2 otherwise $S \cup \{k\} - \{j\}$ could not be on-time for any $j \in (S \cup \{k\}) \cap E1$.

$$\sum_{j \in S2 \cap E1} t(j) = \sum_{j \in (S \cup \{k\}) \cap E1} t(j) > r2$$

$$\Rightarrow \quad I(S2) = 0.$$

It remains to show $I(S1) > 0$.

If not, $I(S1) = 0$

$$\Rightarrow \quad F(S1) = \sum_{j \in S1} t(j)$$

$$< \sum_{j \in S2} t(j) \qquad \text{(since } t(x) < t(y)\text{)}$$

$$= F(S2) \qquad \text{(since } I(S2) = 0\text{)}.$$

This contradicts the given condition, $F(S1) > F(S2)$. ∎

Let S be a set of jobs having the same release time and S* an optimal on-time subset of S obtained by Moore's algorithm [10]. It has been shown in Sidney [11] that S* has the following properties:

(1) S* is on-time.

(2) S* has maximal number of jobs among all on-time subsets of S.

(3) Suppose U is a subset of S* of size q obtained by eliminating those $|S^*|-q$ jobs from S* which have largest execution times amongst the jobs in S*. Then U is an on-time set and has least execution time among all on-

time subsets of S consisting of the same number of jobs.

Let M'(k) and M"(k) be respectively the optimal on-time subsets of $N(k) \cap E1$ and $N(k) \cap E2$ obtained separately by Moore's algorithm [10]. In particular, the above three properties also hold for M'(k) and M"(k).

We now define a particular on-time subset M(k) of N(k), with non-zero idle time, which is the union of M"(k) and a subset of M'(k).

Suppose Q'(k) is a subset of M'(k) obtained by successively eliminating those jobs which have largest execution times amongst the jobs in M'(k) until the total execution time of Q'(k) is <u>just</u> less than r2.

Let

$$M(k) = Q'(k) \cup M''(k)$$

This particular subset of N(k) will be shown in Lemma 3.1.2 to be better than any other on-time subset of N(k) with non-zero idle time. Thus in the search for an optimal schedule, all on-time subsets of N(k) with non-zero idle time can be replaced by M(k).

<u>Lemma 3.1.2</u>

Let S be an on-time subset of N(k) such that

$$I(S) > 0 \quad (\text{i.e.} \quad \sum_{j \in S \cap E1} t(j) < r2).$$

Then

M(k) is better than S.

i.e. for any T, Late[M(k)$\sqcup$T] $\leq$ Late[S$\sqcup$T] + |M(k)| - |S|

Proof.

M(k) = Q'(k)$\sqcup$M"(k) .

To simplify the notations, let us denote M(k), Q'(k), M"(k) and M'(k) by M, Q', M" and M' respectively.

Suppose Late[S$\sqcup$T] = v and the late jobs are

{w1, w2, ..., wv}.

Let

A1 = {w1, ..., wv}$\sqcap$(S$\sqcap$E1)

A2 = {w1, ..., wv}$\sqcap$(S$\sqcap$E2)

A3 = {w1, ..., wv}$\sqcap$T

(N.B. A1, A2 or A3 may be empty)

Obviously,

A1$\sqcup$A2$\sqcup$A3 = {w1, w2, ..., wv}.

Let

S1 = S$\sqcap$E1-A1,

S2 = S$\sqcap$E2-A2.

Then

S$\sqcup$T-{w1, ..., wv} = S1$\sqcup$S2$\sqcup$(T-A3) is on-time.

We are going to construct a subset L of M$\sqcup$T of size (v + |M| - |S|) such that (M$\sqcup$T)-L is on-time. By applying Property (3) to M' with U=Q' and the fact that total execution time of Q' is <u>just</u> less than r2, and also the fact

that $S \cap E1$ is completed before r2, we have

$$|Q'| \geq |S \cap E1|.$$

By Property (2) of M", we have

$$|M"| \geq |S \cap E2|.$$

Let

B' be the subset of Q' obtained by eliminating $(|Q'|-|S1|)$ jobs from Q' which have largest execution times amongst the jobs in Q',

B" be the subset of M" obtained by eliminating $(|M"|-|S2|)$ jobs from M" which have largest execution times amongst the jobs in M".

Now both B' and S1 are on-time subsets of $N(k) \cap E1$ of the same size; so are B" and S2 on-time subsets of $N(k) \cap E2$ of the same size.

By Property (3),

$$\sum_{j \in B'} t(j) \leq \sum_{j \in S1} t(j)$$

and

$$\sum_{j \in B"} t(j) \leq \sum_{j \in S2} t(j)$$

Clearly

$$\sum_{j \in B' \cup B"} t(j) \leq \sum_{j \in S1 \cup S2} t(j)$$

and

$$F(B' \cup B") \leq F(S1 \cup S2) .$$

By Lemma 2.2.2,

$$(S1 \cup S2) \cup (T-A3) \text{ on-time}$$

$\Rightarrow$ $(B' \sqcup B'') \sqcup (T-A3)$ on-time.

The total number of jobs which have been eliminated from $M \sqcup T$ in the construction of $(B' \sqcup B'') \sqcup (T-A3)$ is

$$(|Q'|-|S1|) + (|M''|-|S2|) + |A3|$$
$$= (|Q'|-|S \sqcap E1-A1|) + (|M''|-|S \sqcap E2-A2|) + |A3|$$
$$= |M| - |S| + v.$$

Therefore

$$\text{Late}[M(k) \sqcup T] \leq \text{Late}[S \sqcup T] + |M(k)| - |S| . \quad \blacksquare$$

3.2 The Algorithm and Its Correctness

At each iteration, the algorithm keeps "all" on-time subsets which are potential subsets of an optimal solution of the problem.

It is assumed that for each Job j,

$r(j) + t(j) \leq d(j)$ where $r(j) = r1$ or $r2$, $r1 < r2$.

If this were not the case, the problem size can be reduced by eliminating those jobs that cannot be completed on time regardless of the sequence.

Let C(k) be the set of on-time subsets obtained after iteration k of the algorithm.

The algorithm proceeds as follows.

(Recall the jobs {1,2,...,n} are already ordered by

deadlines)

Step 1. Set C(i) = ϕ, i=0,1,...,n; and k = 1 .

Step 2. If k = n+1, stop. The largest element in the set C(n) is an optimal solution.

Step 3. For each element W in C(k-1) DO

If $F(W \sqcup \{k\}) > d(k)$, then

begin

Let

x and y be jobs such that

$t(y) = \max\{t(j): j \in (W \sqcup \{k\}) \cap E1\}$,

$t(x) = \max\{t(j): j \in (W \sqcup \{k\}) \cap E2\}$.

Let

$S = W \sqcup \{k\} - \{x,y\}$.

If both subsets $S \sqcup \{x\}$ and $S \sqcup \{y\}$ are on-time, then

begin

Let C be the subset with shorter finishing time

(If both subsets have the same finishing time, choose the one with shorter execution time. If they have the same execution time as well, choose $S \sqcup \{y\}$);

If C has no idle time, add C to C(k);

end

else_if the on-time subset has no idle time,
add it to C(k);

end

else_if $W \sqcup \{k\}$ has no idle time, add it to C(k).

Step 4. Add M(k) to C(k).

Step 5. Increment k by 1. Return to step 2.

The following theorem shows the correctness of the algorithm.

Theorem 3.2.1

At the end of iteration k of the algorithm, the set C(k) has the following properties:

(1) There exists $W \in C(k)$ such that W has maximal number of jobs among all on-time subsets of N(k).

(2) For any set T of jobs with deadlines ≥ deadline of any job in N(k) and $T \cap N(k) = \phi$, there exists $W' \in C(k)$ such that the size of an optimal on-time subset of $W' \sqcup T$ equals to that of $N(k) \sqcup T$,
i.e. $Late[W' \sqcup T] + k - |W'| = Late[N(k) \sqcup T]$.

Proof : (induction on k)

The conclusions are true for k = 1, 2.

Assume the conclusions are true for C(1), C(2), ..., C(k-1).

Conclusion (1).

The maximal size of all on-time subsets of N(k) is k-Late[N(k)]. By the induction assumption, there exists W' $\in$ C(k-1) such that

$$\text{Late}[W' \sqcup \{k\}] + (k-1) - |W'| = \text{Late}[N(k-1) \sqcup \{k\}] = \text{Late}[N(k)] . \quad \text{----- (1)}$$

The number of late jobs of W' $\sqcup$ {k} will be either 0 or 1, since W' is on-time. The resultant set C is either W' $\sqcup$ {k} or W' $\sqcup$ {k}-{r} for some r $\in$ W' $\sqcup$ {k}, depending on whether or not W' $\sqcup$ {k} is on-time. C will have maximal number of jobs among all on-time subsets of N(k) according to equality (1).

Note that if I(C) = 0, C $\in$ C(k). If, on the other hand, I(C) > 0, then by the property of M(k) (Lemma 3.1.2), M(k) $\in$ C(k) will have the maximal number of jobs. (Actually, the size of M(k) and C are equal when I(C) > 0). Hence, steps 3 and 4 of the algorithm will produce a maximal on-time set.

Conclusion (2).

Consider in particular the set

$$T' = \{k\} \sqcup T,$$

By the induction assumption, for this T', there exists

$W' \in C(k-1)$ such that

$Late[W' \sqcup T'] + (k-1) - |W'| = Late[N(k-1) \sqcup T']$,

$\Rightarrow \quad Late[W' \sqcup \{k\} \sqcup T] + (k-1) - |W'|$

$= Late[N(k) \sqcup T]$ ----- (2)

Case (i) $W' \sqcup \{k\}$ is on-time

If $W' \sqcup \{k\}$ has no idle time,

then

$W' \sqcup \{k\} \in C(k)$ (by step 3 of the algorithm) .

From equation (2) ,we have

$Late[W' \sqcup \{k\} \sqcup T] + k - |W' \sqcup \{k\}|$

$= Late[(W' \sqcup \{k\}) \sqcup T] + (k-1) - |W'|$

$= Late[N(k) \sqcup T]$.

If $W' \sqcup \{k\}$ has idle time,

then

$M(k) \in C(k)$ is better than $W' \sqcup \{k\}$, (by Lemma 3.1.2).

Therefore

$Late[M(k) \sqcup T] \leq Late[(W' \sqcup \{k\}) \sqcup T] + |M(k)| - |W' \sqcup \{k\}|$.

Rearranging we have

$Late[M(k) \sqcup T] - |M(k)| + |W' \sqcup \{k\}| \leq Late[W' \sqcup \{k\} \sqcup T]$.

Substituting into equation (2), we have

$Late[M(k) \sqcup T] - |M(k)| + |W' \sqcup \{k\}| + (k-1) - |W'|$

$$\leq \text{Late}[N(k) \sqcup T]$$

$$\Rightarrow \quad \text{Late}[M(k) \sqcup T] + k - |M(k)| \leq \text{Late}[N(k) \sqcup T]$$

But

$$\text{Late}[M(k) \sqcup T] + k - |M(k)| \geq \text{Late}[N(k) \sqcup T]$$

therefore

$$\text{Late}[M(k) \sqcup T] + k - |M(k)| = \text{Late}[N(k) \sqcup T] .$$

Hence when $W' \sqcup \{k\}$ is on-time, there exists an on-time subset in $C(k)$, which is either $W' \sqcup \{k\}$ or $M(k)$, satisfying Condition (2) of the theorem.

Case (ii) $W' \sqcup \{k\}$ is not on-time

Then step 3 of the algorithm will produce an on-time subset C of $W' \sqcup \{k\}$ consisting of $|W' \sqcup \{k\}|-1$ jobs. By Theorem 2.3.2, C is better than any on-time subset which result from deleting more than one job from $W' \sqcup \{k\}$.

Using the same notations as the algorithm, from step 3, we have,

either

(a) only one of the subsets $S \sqcup \{x\}$, $x \in E2$; and $S \sqcup \{y\}$, $y \in E1$ is on-time.

Let

$C = S \sqcup \{x\}$ or $S \sqcup \{y\}$, depending on which one is on-time.

In this case, C has the least execution time as well as

the least finishing time among all on-time subsets of $W' \sqcup \{k\}$ consisting of the same number of jobs. By Theorem 2.2.3, C is better than any on-time subset of $W' \sqcup \{k\}$ consisting of the same number of jobs.

Thus

$$\text{Late}[(W' \sqcup \{k\}) \sqcup T] = \text{Late}[C \sqcup T] + 1 .$$

From Equation (2) we have

$$\text{Late}[C \sqcup T] + 1 + (k-1) - |W'| = \text{Late}[N(k) \sqcup T]$$

$$\Rightarrow \quad \text{Late}[C \sqcup T] + k - |C| = \text{Late}[N(k) \sqcup T].$$

Furthermore, if C has no idle time then $C \in C(k)$.
This yields the desired result.

However, if C has an idle time, then $M(k) \in C(k)$ is better than C (by Lemma 3.1.2). By the same technique as in Case (i), we can show that M(k) satisfies Condition (2), i.e.

$$\text{Late}[M(k) \sqcup T] + k - |M(k)| = \text{Late}[N(k) \sqcup T] .$$

We remark that C is in fact as good as M(k).

or

(b) Both subsets $S \sqcup \{x\}$, $x \in E2$; and $S \sqcup \{y\}$, $y \in E1$ are on-time.

Then step 3 of the algorithm will give the subset C which has least finishing time among all on-time subsets of $W' \sqcup \{k\}$ consisting of the same number of jobs.

$C = S \sqcup \{r\}$ where $r = y$ or x.

If $r = x$ $(x \in E2)$, then C has the least execution time as well. Applying the same method as (a), we have

$$Late[C \sqcup T] + k - |C| = Late[N(k) \sqcup T]$$

(if $I(C) = 0$ and so $C \in C(k)$)

or

$$Late[M(k) \sqcup T] + k - |M(k)| = Late[N(k) \sqcup T]$$

(if $I(C) > 0$).

If $r = y$ $(y \in E1)$,

and if $t(y) \leq t(x)$, then

C again has the least execution time. The same technique applies here and the result follows.

Otherwise (when $t(y) > t(x)$) let

$C' = S \sqcup \{x\}$ $x \in E2$.

By Theorem 2.2.3, C' and C are respectively better than any on-time subset which result from deleting a E1 and a E2 job from $W' \sqcup \{k\}$.

Since $t(y) > t(x)$ and C has the least finishing time, we have,

$$\sum_{j \in C'} t(j) < \sum_{j \in C} t(j)$$

but $F(C') > F(C)$.

By Lemma 3.1.1, C' has idle time but C does not.

This implies that C belongs to C(k).

From Lemma 3.1.2, M(k) is better than C'. Thus we have,

$$\text{Late}[W' \sqcup \{k\} \sqcup T] = 1 + \min\{\text{Late}[M(k) \sqcup T], \text{Late}[C \sqcup T]\}$$

where M(k) and $C \in C(k)$.

By a similar technique to that used in (a), we can show that either C or M(k) satisfies Condition (2) of the theorem.

This completes the proof. ∎

3.3 Complexity of the Algorithm

The data structures to facilitate the constructions of C(k) are heaps [1]. A heap is a particular binary tree with the largest element at the root. A heap of k nodes can be built in O(k) time; it can also be updated in O(log k) time upon deletion of the root or insertion of a new node.

For each element W in C(k), a pair of heaps is maintained; each node is a job and its position in the heap is determined by its execution time. One heap is for the set of E1 jobs in W and the other for the set of E2 jobs in W. An information triple is stored for each W. They are F(W), total execution time of E1 jobs in W, and |W|.

Consider iteration k, at step 3, there are $|C(k-1)| \leq$

k-1 iterations. For each element W in C(k-1), it requires updating the pair of heaps of W and the information triple. These can be done in O(log k) time; and so step 3 takes O(k log k) time. Other steps clearly take no more than O(k log k) time.

There are n iterations. Therefore the time complexity of the algorithm is of order

$$\sum_{k=1}^{n} (k \log k) \quad \text{or} \quad O(n^2 \log n) \text{ time.}$$

■

Chapter IV

Particular Cases and Heuristic Algorithm

We have found an optimal algorithm for the case of two release times. We shall try to solve the very hard problem of m release times, where $m \leq n$. (n is the number of jobs).

In this chapter we study the problem for the case of arbitrary release times. In Section 4.1 we outline the general model and define the D-schedule for the general case. In Section 4.2 we study a particular case where the subset of jobs released at times later than the earliest release time is given on-time and we present an O(n log n) optimal algorithm to schedule these jobs. We further extend this algorithm in Section 4.3 to a heuristic algorithm for the case of arbitrary release times and show that the number of late jobs produced by this heuristic algorithm is no more than m times the optimal. We also construct an example which achieves the bound for m=2.

4.1 General Model

Consider a set of jobs $J = \{1, 2, \ldots, n\}$ with arbitrary release times $r(i)$, $i=1,2,\ldots, n$.

Let m be the number of distinct release times, $m \leq n$;

$r1, r2, \ldots, rm$ be the m distinct release times;

and Ei be the subset of jobs released at ri, $1 \leq i \leq m$.

Without loss of generality, we assume

$r1 < r2 < \ldots < rm$

and $d(1) \leq d(2) \leq \ldots \leq d(n)$.

In analogy to the case of two release times, we have the following definition of D-schedule for m release times.

Definition 4.1.1

A D-schedule of a subset S of J is a schedule which satisfies the following conditions.

(1) All jobs in $S \cap Ei$ are executed according to the order of non-decreasing deadlines, for i=1, ..., m. If d(i) = d(j) and i < j then Job i is executed before Job j.

(2) The jobs (including,if any, the remaining portions of jobs which have been preempted at some release times) scheduled between any two release times are executed according to the order of non-decreasing deadlines. If d(i) = d(j) and i < j then Job i is executed before Job j. ■

Let S be a subset of J (not necessarily on-time) with the latest job released at time rp ($1 \leq p \leq m$). The finishing time F(S) of the D-schedule of S can be computed by the following equation.

$$F(S) = \sum_{j \in S} t(j) + \sum_{j=2}^{p} Ij(S) \quad \text{--------} \quad (1)$$

where Ij(S) is the idle period right before rj (Figure 4.1.1) when S is arranged by D-schedule. Ij(S) is non-zero

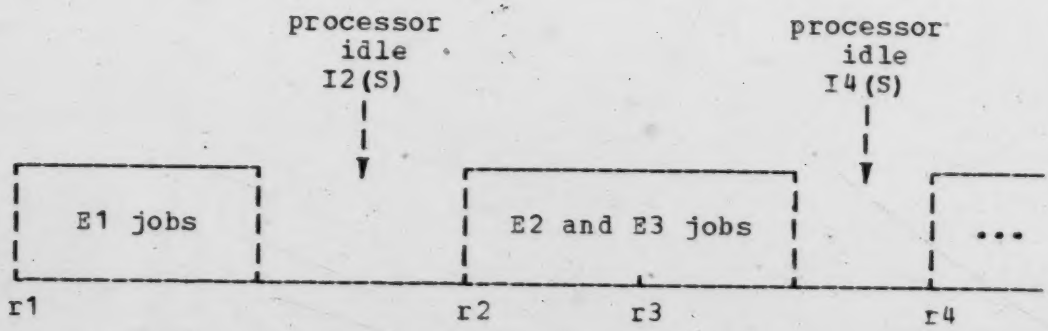


Figure 4.1.1 : A D-schedule for m Release Times.

if all the jobs with release times earlier than rj are completed execution before rj. They can be computed by the following formulas.

$I1(S) = 0$

$$Ij(S) = \max\{0, rj - (\sum_{j \in S \cap [E1 \cup \ldots \cup Ej']} t(j) + \sum_{i=1}^{j-1} Ii(S))\}$$

where $j'=j-1$.

It can be shown that Lemma 2.1.2 and Theorem 2.1.3 of Chapter 2 are also true for the case of m release times. We now restate them for the general case here.

Lemma 4.1.1

S is an on-time subset of J if and only if the D-schedule of S is on-time.

Theorem 4.1.2

$S = \{1, 2, \ldots, p\}$ is an on-time subset of J

if and only if

$F(\{1, 2, \ldots, i\}) \leq d(i)$ $i=1, \ldots, p.$

(N.B. $d(1) \leq d(2) \leq \ldots \leq d(p)$)

4.2 An Optimal Case

Consider a particular case where the subset of jobs released at times later than r1 is given on-time. i.e. $E2 \cup E3 \cup \ldots \cup Em$ is on-time. A simple algorithm of deleting the largest job whenever a subset of jobs is not on-time will produce an optimal on-time subset.

We first present the algorithm and then prove it to be correct in Theorem 4.2.1.

Algorithm-1 (recall the jobs are already ordered by deadlines)

step 1. Set $C(0) = \phi$ and $k=1$.

step 2. If $k=n+1$, stop. $C(n)$ is an optimal set.

step 3. If $F(C(k-1) \cup \{k\}) \leq d(k)$ set $C(k) = C(k-1) \cup \{k\}$ otherwise let x be a job such that $t(x) = \max\{t(j): j \in C(k-1) \cup \{k\}\}$ and set $C(k) = C(k-1) \cup \{k\} - \{x\}$.

step 4. Increment k by 1. Return to step 2. ∎

Algorithm-1 is analogous to that of Sidney [11]. It is an $O(n \log n)$ algorithm if data structures similar to those for the algorithm of Chapter 3 are used.

Theorem 4.2.1

Let $N(k) = \{1, 2, \ldots, k\}$ be the set of the first k jobs of J, $k \geq 1$. (N.B. J is already ordered by non-decreasing deadlines). At the end of iteration k of Algorithm-1, the set $C(k)$ has the following properties :

(1) $C(k)$ is on-time.

(2) $C(k)$ has maximal number of jobs among all on-time subsets of $N(k)$.

(3) Suppose W is a subset of $C(k)$ of size q obtained by eliminating those $|C(k)|-q$ jobs from $C(k)$ which have largest execution time amongst the jobs in $C(k)$. Then W

is an on-time set and has least execution time among all on-time subsets of N(k) consisting of the same number of jobs.

(The statement of this theorem is analogous to the one in Sidney [11]).

Proof (by induction on k)

The conclusions are true for k=1.

Assume the conclusions are true for 1, 2, .., k-1.

Conclusion (1)

(i) $F(C(k-1) \sqcup \{k\}) \leq d(k)$.

By Theorem 4.1.2, $C(k) = C(k-1) \sqcup \{k\}$ is on-time.

(ii) $F(C(k-1) \sqcup \{k\}) > d(k)$.

If x=k in step 3 of Algorithm-1, then C(k) = C(k-1) is on-time.

Otherwise $C(k) = C(k-1) \sqcup \{k\} - \{x\}$.

If the D-schedule of C(k) has idle times, then by the fact that $E2 \sqcup \ldots \sqcup Em$ is on-time, it can be shown easily that C(k) is on-time.

else
$$F(C(k)) = \sum_{j \in C(k)} t(j) = \sum_{j \in C(k-1)} t(j) + t(k) - t(x)$$
$$\leq \sum_{j \in C(k-1)} t(j)$$
$$\leq F(C(k-1))$$
$$\leq d(k-1)$$
$$\leq d(k).$$

and every job in C(k-1)-{r} will be completed at least as

early as before.

Hence C(k) is on-time.

Conclusion (2)

By induction assumptions, the maximal number of jobs among all on-time subsets of N(k) will be either |C(k-1)| or |C(k-1)|+1.

If some maximal on-time subset V of N(k) does consist of |C(k-1)|+1 elements, then V must contain job k. Hence an on-time subset V of N(k) consisting of |C(k-1)|+1 elements exists if and only if there exists an on-time subset V' of N(k-1) consisting of |C(k-1)| elements having the property that $F(V' \sqcup \{k\}) \leq d(k)$.

By the induction assumptions on C(k-1), it can be proved that such a V' exists if and only if $F(C(k-1) \sqcup \{k\}) \leq d(k)$.

Hence C(k) defined at step 3 of Algorithm-1 is maximal.

Conclusion (3)

Observe that $C(k) = C(k-1) \sqcup \{k\}$ if $C(k-1) \sqcup \{k\}$ is on-time. Otherwise, C(k) consists of |C(k-1)| smallest jobs of $C(k-1) \sqcup \{k\}$.

Furthermore Conclusion (3) of this theorem defines W to be that subset of C(k) consisting of q smallest jobs in C(k). These observations yield the desired result. ∎

4.3 Heuristic Algorithm

We have just shown that Algorithm-1 gives an optimal on-time subset for the particular case where the subset of jobs released at times later than the earliest release times is given on-time. Making use of Algorithm-1 , we now present a heuristic algorithm for the general problem of arbitrary release times (the model of Section 4.1). The number of late jobs produced by this heuristic algorithm is no more than m times the optimal.

The heuristic algorithm first applies Algorithm-1 on Em to find an on-time subset and then iteratively (j=m-1 to 1 step -1) applies Algorithm-1 on Ej and the on-time subset of the jobs released after rj which is obtained in the previous iteration. ∎

The heuristic algorithm is of O(mn log n) since Algorithm 1 can be implemented in O(n log n) time.

At each iteration, Algorithm-1 works on a subset of J. By Theorem 4.2.1, it produces a maximal on-time subset for the set of jobs it works on. Clearly it gives no more than Late[J] late jobs at each iteration. There are altogether m iterations, therefore the number of late jobs given by the heuristic algorithm is not more than m*Late[J]. ∎

We claim that the bound is the lowest possible for m=2. Consider the following example for m=2 with r1=0 and r2=1,

where

E2 (2v jobs with r(i)=1)

	job								
	1	2	3	4	...	2j-1		2j	... 2v
t(i)	p	p+1	2p	2p+1	...	$2^{j-1}*p$		$2^{j-1}*p+1$	...
d(i)	p+1	2p+1	3p+1	5p+1	...	$(2^{j}-1)*p+1$		$(3*2^{j-1}-1)*p+1$	...

j = 1, 2, ..., v.

p is an integer > v + 1

E1 (v jobs with r(i)=0)

	job						
	1	2	3	...	j	...	v
t(i)	p-1	p-1	2p-1	...	$2^{j-2}*p-1$	...	
d(i)	p	3p	7p	...	$(2^{j}-1)*p$	...	

j=2, 3, ..., v.

The heuristic algorithm first applies Algorithm-1 on E2 to find an optimal on-time subset E2* for E2 and then applies Algorithm-1 on E1⊔E2* to find an optimal on-time subset for E1⊔E2*.

The finishing time of the first k cdd numbered jobs of E2 is :

$$1 + \sum_{j=1}^{k} t(2j-1) = 1 + p^* \sum_{j=1}^{k} 2^{j-1}$$

$$= (2^k - 1)*p + 1$$

$$= d(2k-1) \qquad k=1, \ldots, v.$$

The finishing time of the first k odd numbered jobs and the next even numbered job of E2 is :

$$1 + \sum_{j=1}^{k} t(2j-1) + t(2k) = (2^k - 1)*p + 1 + (2^{k-1}*p + 1)$$

$$= (3*2^{k-1} - 1)*p + 2$$

$$> d(2k) \qquad k=1, \ldots, v.$$

Algorithm-1 will delete the largest job which is the (2k)th job of E2. Eventually Algorithm-1 will produce an optimal on-time subset

E2* = {odd numbered jobs of E2}.

Similarly it can be shown that the optimal on-time subset of E1$\cup$E2* produced by Algorithm-1 is exactly E1. The total number of late jobs given by the heuristic algorithm is 2v.

But E1$\cup$ { even numbered jobs of E2 } is an optimal on-time subset for E1$\cup$E2 and the optimal number of late jobs for the problem is v.

Thus the number of late jobs given by the heuristic algorithm is exactly twice the optimal.

CHAPTER V

Conclusion

This thesis deals with the problem of scheduling a set of jobs with given execution times, release times and deadlines in order to minimize the number of late jobs. Moore [10] has found a simple algorithm for the problem on one processor with jobs released at the same time. Unfortunately, the problem becomes rather involved when the job release times are different.

In the thesis, we have analysed the problem for two batches of jobs of different release times in great detail. We have proved in Appendix A that the problem is NP-complete when preemption is not allowed. The algorithm presented in Chapter III has been shown to produce an optimal preemptive schedule in $O(n^2 \log n)$ time. A more efficient optimal preemptive algorithm of time complexity $O(n \log n)$ has been found for a particular case, where the jobs released at times later than the earliest release time is given on-time. A heuristic algorithm for the problem with m release times has also been presented, where m is less than or equal to the size of the problem. It has been proved that the number of late jobs produced by this algorithm is no more than m times the optimal. The bound of the heuristic algorithm for $m=2$ has been shown to be the lowest possible. In addition,

we have defined a special schedule, the D-schedule, which gives an on-time schedule whenever one exists.

Whether the problem for m release times is NP-complete when preemption is allowed, is still unknown. However, it is hoped that the optimal algorithm given in Chapter III can be generalized to the case of m release times.

References

[1] Aho, A.V., Hopcroft, J.E., and Ullman, J.D., "The Design and Analysis of Computer Algorithms", Addison Wesley, 1974.

[2] Baker, K.R, "Introduction to Sequencing and Scheduling", John Wiley & Sons, 1974.

[3] Blazewicz, J., "Simple Algorithms For Multiprocessor Scheduling to Meet Deadlines", Information Processing Letters Vol. 6, No. 5 (Oct. 1977) pp. 162-164.

[4] Coffman, E.G., "Computer and Job Shop Scheduling Theory", Wiley, New York, 1976.

[5] Garey, M.R., and Johnson D.S., "Scheduling Tasks with Nonuniform Deadlines on Two Processors", JACM 23 (1976) pp. 461-467.

[6] Garey, M.R., and Johnson, D.S., "Scheduling Tasks With Start-times and Deadlines", SIAM J. Comput. Vol.6 No. 3, (Sept.1977) pp. 416-426.

[7] Jackson, J.K., "Scheduling a Production Line to Minimize Maximum Tardiness", Research Report 43, Management Science Research Project, UCLA, January, 1955.

[8] Karp, R.M., "Reducibility Among Combinational Problems", Complexity of Computer Computation, R.E. Miller and J.W. Thatcher (Ed. 3), Plenum Press, New York, 1972, pp. 85-104.

[9] Kise, H., Ibaraki, T., and Mine, H., "A Solvable Case of the One-Machine Scheduling Problem with Ready and Due Times", Operations Research Vol.26 No.1, January-February 1978.

[10] Moore, J.M., "An n Job, One Machine Sequencing Algorithm For Minimizing the Number of Late Jobs", Management Science, Vol.15 No.1 (Sept.1968) pp. 102-109.

[11] Sidney, J.E., "An Extension of Moore's Due-date Algorithm", Symposium on the Theory of Scheduling and its Applications, S.M. Elmaghraby, Ed., Springer Verlag New York (1973) pp. 393-398.

[12] Simons, B., "A Fast Algorithm for Single Processor Scheduling", Symposium on Foundations of Computer Science, October, 1978.

Appendix A

In showing the NP-Completeness result, we shall make use of the following known NP-Complete problem.

Partition : A multiset $S = \{a(1), a(2), \ldots, a(n)\}$ is said to have a partition if there exists a subset, u, of the indices $\{1,2,\ldots,n\}$ such that $\sum_{i \in u} a(i) = \left(\sum_{i=1}^{n} a(i) \right)/2$.

The partition problem is that of determining for an arbitrary multiset S whether it has a partition. The a(i)'s may be assumed to be integers.

Theorem A

For a given set of jobs with arbitrary execution times t(i), arbitrary deadlines d(i) and release times r(i)=r1 or r2 where $r1 \leq r2$, the problem of determining whether there exists an on-time non-preemptive schedule on a single processor is NP-complete.

Proof :

The problem belongs to NP [6]. We shall give a reduction of the PARTITION problem to it. From the partition problem $S = \{a(1), a(2), \ldots, a(n)\}$ construct the following non-preemptive scheduling problem with n+1 jobs :

Job i : $r(i)=0$, $t(i)=a(i)$, $d(i)=A+1$; $1 \leq i \leq n$

Job n+1 : $r(n+1)=A/2$, $t(n+1)=1$, $d(n+1)=A/2 + 1$

where A is the sum of all a(i)'s.

Clearly this scheduling problem can be constructed in polynomial time. We now show that there exists an on-time non-preemptive schedule if and only if S has a partition.

(a) If S has a partition u then there is an on-time non-preemptive schedule. One such schedule is shown in Figure A.

(b) To prove the other direction, suppose there is an on-time non-preemptive schedule. It is easy to see that in any on-time schedule, Job n+1 must be scheduled at time A/2, and the schedule cannot have idle time. Therefore we can let the set u from the partition problem be the set of indices of the jobs that are scheduled before time A/2 or after time (A/2 + 1).

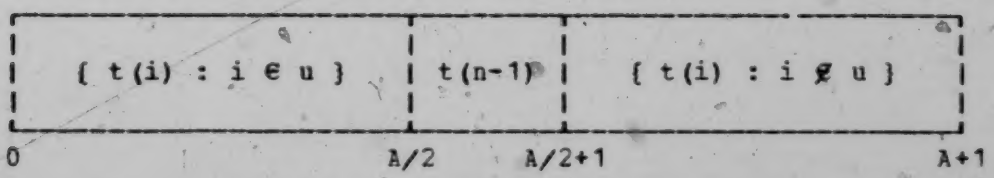


Figure A : Partition of Jobs - A Non-preemptive Schedule